Naive Bayes Homework

硕-8089 3118111009 许林松

I. QA

1

Abstractly, naive Bayes is a conditional probability model: given a problem instance to be classified, represented by a vector $\mathbf{x} = (x_1, \ldots, x_n)$ representing some n features (independent variables), it assigns to this instance probabilities :

$$p(C_k \mid x_1, \ldots, x_n)$$

for each of K possible outcomes or "classes" C_k

The problem with the above formulation is that if the number of features n is large or if a feature can take on a large number of values, then basing such a model on probability tables is infeasible. We therefore reformulate the model to make it more tractable. Using Bayes' theorem, the conditional probability can be decomposed as:

$$p(C_k \mid \mathbf{x}) = \frac{p(C_k)\ p(\mathbf{x} \mid C_k)}{p(\mathbf{x})}$$

In plain English, using Bayesian probability terminology, the above equation can be written as:

$$\text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}$$

In practice, there is interest only in the numerator of that fraction, because the denominator does not depend on C and the values of the features x_i are given, so that the denominator is effectively constant.

The numerator is equivalent to the joint probability model:

$$p(C_k, x_1, \ldots, x_n)$$

Now the "naive" conditional independence assumptions come into play: assume that each feature x_i is conditionally statistical independence|independent of every other feature x_j for $j \neq i$, given the category C_k. This means that:

$$p(x_i \mid x_{i+1}, \ldots, x_n, C_k) = p(x_i \mid C_k)$$

Thus, the joint model can be expressed as:

$$\begin{aligned} p(C_k \mid x_1, \ldots, x_n) &\propto p(C_k, x_1, \ldots, x_n) \\ &= p(C_k)\ p(x_1 \mid C_k)\ p(x_2 \mid C_k)\ p(x_3 \mid C_k)\ \cdots \\ &= p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)\,. \end{aligned}$$

Where $\propto$ denotes Proportionality.

This means that under the above independence assumptions, the conditional distribution over the class variable C is:

$$p(C_k \mid x_1, \ldots, x_n) = \frac{1}{Z} p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)$$

where the evidence

$$Z = p(\mathbf{x}) = \sum_k p(C_k)\ p(\mathbf{x} \mid C_k)$$

is a scaling factor dependent only on

$$x_1, \ldots, x_n$$

, that is, a constant if the values of the feature variables are known.

Constructing a classifier from the probability model

The discussion so far has derived the independent feature model, that is, the naive Bayes probability model. The naive Bayes classifier combines this model with a decision rule. One common rule is to pick the hypothesis that is most probable; this is known as the "maximum a posteriori" or "MAP" decision rule. The corresponding classifier, a Bayes classifier, is the function that assigns a class label $\hat{y} = C_k$ for some k as follows:

$$\hat{y} = \underset{k \in \{1, \ldots, K\}}{\operatorname{argmax}}\ p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)$$

2

In simple terms, a naive Bayes classifier assumes that the presence (or absence) of a particular feature of a class is unrelated to the presence (or absence) of

any other feature, given the class variable. Even if these features depend on each other or upon the existence of the other features, a naive Bayes classifier considers all of these properties to independently contribute to the probability.

3

Supervised Machine Learning

The majority of practical machine learning uses supervised learning.

Supervised learning is where you have input variables x and an output variable y and you use an algorithm to learn the mapping function from the input to the output.

$$y = f(x)$$

The goal is to approximate the mapping function so well that when you have new input data x that you can predict the output variables y for that data.

It is called supervised learning because the process of an algorithm learning from the training dataset can be thought of as a teacher supervising the learning process. We know the correct answers, the algorithm iteratively makes predictions on the training data and is corrected by the teacher. Learning stops when the algorithm achieves an acceptable level of performance.

Supervised learning problems can be further grouped into regression and classification problems.

Unsupervised Machine Learning

Unsupervised learning is where you only have input data (X) and no corresponding output variables.

The goal for unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data.

These are called unsupervised learning because unlike supervised learning above there is no correct answers and there is no teacher. Algorithms are left to their own devises to discover and present the interesting structure in the data.

Unsupervised learning problems can be further grouped into clustering and association problems.

4

What?

In machine learning, overfitting occurs when a learning model customizes itself too much to describe the relationship between training data and the labels. Overfitting tends to make the model very complex by having too many parameters. By doing this, it loses its generalization power, which leads to poor performance on new data.

Why?

The reason this happens is because we use different criteria to train the model and then test its efficiency. As we know, a model is trained by maximizing its accuracy on the training dataset. But its performance is determined on its ability to perform well on unknown data. In this situation, overfitting occurs when our model tries to memorize the training data as opposed to try to generalize from patterns observed in the training data.

For example, let' s say if the number of parameters in our model is greater than the number of datapoints in our training dataset. In this case, a learning model can predict the output for training data by simply memorizing the entire training dataset. But as you can imagine, such model will fail drastically when dealing with unknown data.

How?

One way to avoid overfitting is to use a lot of data. The main reason overfitting happens is because you have a small dataset and you try to learn from it. The algorithm will have greater control over this small dataset and it will make sure it satisfies all the datapoints exactly. But if you have a large number of datapoints, then the algorithm is forced to generalize and come up with a good model that suits most of the points.

We don' t have the luxury of gathering a large database all the time. Sometimes we are limited to a small database and we are forced to come up with a model based on that. In these situations, we use a technique called cross validation. What it does it that it splits the dataset into training and testing datasets. Only the datapoints in the training dataset are used to come up with the model and the testing dataset is used to test how good the model is. This is repeated with different partitions of training and testing datasets. This method gives a fairly good estimate of the underlying model because we are testing it on different partitions to

generalize it as much as possible.

5

In short, Artificial Intelligence is the broader concept of machines being able to carry out tasks in a way that we would consider "smart". And, Machine Learning is a current application of AI based around the idea that we should really just be able to give machines access to data and let them learn for themselves.

6

$$\begin{aligned} P(A \mid bad) &= \frac{P(A, bad)}{P(bad)} \\ &= \frac{P(bad \mid A)P(A)}{P(bad \mid A)P(A) + P(bad \mid B)P(B)} \\ &= \frac{0.3 \times 0.03}{0.3 \times 0.03 + 0.7 \times 0.01} \\ &= \frac{9}{16} \end{aligned}$$

7

$$\begin{aligned} & P(yes \mid 985, master, C++) \\ &= \frac{P(985, master, C++ \mid yes)P(yes)}{P(985, master, C++)} \\ &\propto P(yes)P(985 \mid yes)P(master \mid yes)P(C++ \mid yes) \\ &= \frac{6}{10} \times \frac{4}{6} \times \frac{2}{6} \times \frac{1}{6} \\ &= \frac{1}{45} \end{aligned}$$

$$\begin{aligned} & P(no \mid 985, master, C++) \\ &= \frac{P(985, master, C++ \mid no)P(refuse)}{P(985, master, C++)} \\ &\propto P(no)P(985 \mid offer)P(master \mid no)P(C++ \mid no) \\ &= \frac{4}{10} \times \frac{1}{4} \times \frac{2}{4} \times \frac{3}{4} \\ &= \frac{3}{80} \end{aligned}$$

Because $\frac{1}{45} < \frac{3}{80}$, so he can't get the offer.

II. Multiple choice question

- D
- C
- A
- A,B
- A
- A,B
- B
- A
- A,B,C
- C
- D

III. Programming

```
import numpy as np
import pandas as pd
import random
from sklearn import model_selection, naive_bayes
X = []
data = pd.read_csv('career_data.csv')
for i in range(data.shape[0]):
    tmp = []
    if data.iloc[i]['985'] == 'Yes':
        tmp.append(1)
    else:
        tmp.append(-1)
    if data.iloc[i]['education'] == 'bachlor':
        tmp.append(1)
    elif data.iloc[i]['education'] == 'master':
        tmp.append(0)
    else:
        tmp.append(-1)
    if data.iloc[i]['skill'] == 'C++':
        tmp.append(1)
    else:
        tmp.append(-1)
    X.append(tmp)
    if data.iloc[i]['enrolled'] == 'No':
        tmp.append(0)
    else:
        tmp.append(1)
X = np.array(X)
for i in range(10):
    a = model_selection.train_test_split(X[:,:3],
    X_train, X_test, y_train, y_test = a
    cls = naive_bayes.GaussianNB()
    cls.fit(X_train, y_train)
    print('GaussianNB Testing-%d Score: %.2f'
% (i,cls.score(X_test, y_test)))
```

The result is:

$GaussianNB \quad Testing-0 \quad Score: 0.50$

$GaussianNB \quad Testing-1 \quad Score: 1.00$

$GaussianNB \quad Testing-2 \quad Score: 0.50$

$GaussianNB \quad Testing-3 \quad Score: 1.00$

$GaussianNB \quad Testing-4 \quad Score: 0.50$

$GaussianNB \quad Testing-5 \quad Score: 0.00$

$GaussianNB \quad Testing-6 \quad Score: 1.00$

$GaussianNB \quad Testing-7 \quad Score: 0.50$

$GaussianNB \quad Testing-8 \quad Score: 0.00$

$GaussianNB \quad Testing-9 \quad Score: 1.00$

IV. Apriori

This section I wrote mainly refers to 《Machine Learning in Action》 .

Association analysis is the task of finding interesting relationships in large datasets. These interesting relationships can take two forms: frequent item sets or association rules. *Frequent item sets* are a collection of items that frequently occur together. The second way to view interesting relationships is association rules. *Association rules* suggest that a strong relationship exists between two items.

Apriori A priori means "from before" in Latin. When defining a problem, it' s common to state prior knowledge, or assumptions. This is written as "a priori." In Bayesian statistics, it' s common to make inferences conditional upon this a priori knowledge. A priori knowledge can come from domain knowledge, previous measurements, and so on.

Finding frequent itemsets with the Apriori algorithm

```
import numpy as np
def loadDataSet():
    return [[1,3,4],[2,3,5],[1,2,3,5],[2,5]]
def createC1(dataSet):
    C1 = []
    for transaction in dataSet:
        for item in transaction:
            if not [item] in C1:
                C1.append([item])
    C1.sort()
    return list(map(frozenset, C1))
def scanD(D, Ck, minSupport):
    ssCnt = {}
    for tid in D:
        for can in Ck:
            if can.issubset(tid):
                if can not in ssCnt: ssCnt[can]=1
                else: ssCnt[can] += 1
    numItems = float(len(D))
    retList = []
    supportData = {}
    for key in ssCnt:
        support = ssCnt[key]/numItems
        if support >= minSupport:
            retList.insert(0,key)
        supportData[key] = support
    return retList, supportData
def aprioriGen(Lk, k): #creates Ck
    retList = []
    lenLk = len(Lk)
    for i in range(lenLk):
        for j in range(i+1, lenLk):
            L1 = list(Lk[i])[:k-2]
            L2 = list(Lk[j])[:k-2]
            L1.sort(); L2.sort()
            if L1==L2:
                retList.append(Lk[i] | Lk[j])
    return retList
def apriori(dataSet, minSupport = 0.5):
    C1 = createC1(dataSet)
    D = list(map(set, dataSet))
    L1, supportData = scanD(D, C1, minSupport)
    L = [L1]
    k=2
    while (len(L[k-2]) > 0):
        Ck = aprioriGen(L[k-2], k)
        print(k)
        print(Ck)
        Lk, supK = scanD(D, Ck, minSupport)
        supportData.update(supK)
        L.append(Lk)
        k += 1
    return L, supportData
dataSet = loadDataSet()
```

```
apriori(dataSet)
```

The result is:

([[$frozenset(1), frozenset(3), frozenset(2), frozenset(5)$],
[$frozenset(3,5), frozenset(1,3), frozenset(2,5),$
$frozenset(2,3)$],
[$frozenset(2,3,5)$],
[]],
$frozenset(5) : 0.75,$
$frozenset(3) : 0.75,$
$frozenset(2,3,5) : 0.5,$
$frozenset(1,2) : 0.25,$
$frozenset(1,5) : 0.25,$
$frozenset(3,5) : 0.5,$
$frozenset(4) : 0.25,$
$frozenset(2,3) : 0.5,$
$frozenset(2,5) : 0.75,$
$frozenset(1) : 0.5,$
$frozenset(1,3) : 0.5,$
$frozenset(2) : 0.75$)

Mining association rules from frequent item sets

```
def Rules(L, supp, Conf=0.7):
    big = []
    for i in range(1, len(L)):
        for fr in L[i]:
            H1=[frozenset([t]) for t in fr]
            if(i > 1):
                rFC(fr,H1,supp,big,Conf)
            else:
                calc(fr,H1,supp,big,Conf)
    return big
def calc(fr,H,supp,brl,Conf=0.7):
    prunedH = []
    for c1 in H:
        co=supp[fr]/supp[fr-c1]
        if co >= Conf:
            print (fr-c1,'->',c1,'conf:',co)
            brl.append((fr-c1,c1,co))
            prunedH.append(c1)
    return prunedH
def rFC(fr,H,supp,brl,Conf=0.7):
    m = len(H[0])
    if (len(fr) > (m + 1)):
        Hmp1 = aprioriGen(H, m + 1)
        Hmp1 = calc(fr,Hmp1,supp,brl,Conf)
        if (len(Hmp1) > 1):
            rFC(fr,Hmp1,supp,brl,Conf)
```

The result when the $Conf = 0.5$ is:

$frozenset(5)$	$->$	$frozenset(3)$	$conf:$	0.66
$frozenset(3)$	$->$	$frozenset(5)$	$conf:$	0.66
$frozenset(3)$	$->$	$frozenset(1)$	$conf:$	0.66
$frozenset(1)$	$->$	$frozenset(3)$	$conf:$	1.0
$frozenset(5)$	$->$	$frozenset(2)$	$conf:$	1.0
$frozenset(2)$	$->$	$frozenset(5)$	$conf:$	1.0
$frozenset(3)$	$->$	$frozenset(2)$	$conf:$	0.66
$frozenset(2)$	$->$	$frozenset(3)$	$conf:$	0.66
$frozenset(5)$	$->$	$frozenset(2,3)$	$conf:$	0.66
$frozenset(3)$	$->$	$frozenset(2,5)$	$conf:$	0.66
$frozenset(2)$	$->$	$frozenset(3,5)$	$conf:$	0.66

V. FP-growth

FP-growth builds from Apriori but uses some different techniques to accomplish the same task. That task is finding frequent itemsets or pairs, sets of things that commonly occur together, by storing the dataset in a special structure called an FP-tree. This results in faster execution times than Apriori, com- monly with performance two orders of magnitude better.

Two of the most common ways of looking at things in the dataset are frequent itemsets and association rules. FP-growth allows us to mine data more efficiently. This algorithm does a better job of finding frequent itemsets, but it doesn' t find association rules.

The FP-growth algorithm is faster than Apriori because it requires only two scans of the database, whereas Apriori will scan the dataset to find if a given pattern is frequent or not—Apriori scans the dataset for every potential frequent item. On small datasets, this isn' t a problem, but when you' re dealing with larger datasets,

this will be a prob- lem. The FP-growth algorithm scans the dataset only twice.

Because the the problem of typographic format, I don't write the code on this homework. If you are interested in it, you can find the code I think good from here